Instituto Superior de Engenharia de Coimbra

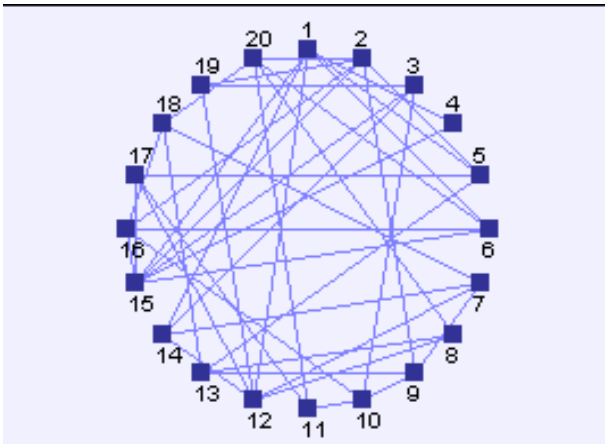
Introdução à Inteligência Artificial

Licenciatura em Engenharia Informática, Engenharia Informática – Pós Laboral e Engenharia Informática – Curso Europeu

2º Ano – 1º semestre 2016/2017

Aulas Laboratoriais

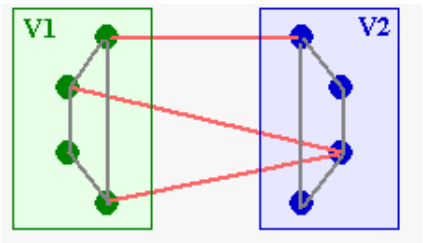
Classes 7 and 8: Local Search



1. Problem: Minimum bisection of a graph

The bisection of a graph is an optimization problem that often arises in real-world situations (for example, scheduling problems or VLSI design). The goal is to divide the vertices in two sets with the same number of elements, minimizing the number of vertices of Set V1 linked to the vertices of Set V2.

This problem can be defined as follows:
- There is a graph G = (V, A) with an even number V of vertices and A arches.
- The goal is to divide the set V into two disjoint sets V1 and V2 with the same cardinality (ie, with the same number of vertices).
- The cost of the bisection is given by the number of arcs making connections between vertices that belong to different clusters (i.e., arcs that connect one vertex V1 to vertex V2).
- The purpose of optimization is to minimize the cost of bisection.



2. Components of optimization algorithms

In this class we will used different local search algorithms to perform optimization.
The following described components are common to all of them.

2.1 Representation

In one instance with N vertices, a solution must indicate which N / 2 nodes belong to set V1 and which N / 2 nodes belong to the set V2.
For that purpose, a binary vector with N positions is used. Each bit of this vector represents a vertex in the graph. A position with the value '0' indicates that the vertex belongs to the set V1, while the value '1' indicates that belongs to V2.

Example: In an instance with 6 vertices, the solution 101100 specifies that the vertices 1, 3 and 4 belong to set V1 and 2, 5 and 6 to set V2.
For solutions to be valid it must be ensured that the number of '0' and '1' is always the same.

2.2 Evaluation and Optimization Goal

The quality of a solution is equal to the cost of the proposed bisection.
The objective is to find a solution where this cost is minimized (minimization problem).

2.3. Local neighborhood

To create a neighbor of a solution the following steps must be performed:
- Choose a P1 position of the binary vector with '0'
- Choose a P2 position of the binary vector with value '1'
- Change the values of the bits in P1 and P2 positions

In practice, this neighborhood chooses a vertex in each of the sets V1 and V2 and swaps them. This neighborhood ensures that the number of '0' and '1' remains the same.

3. Hill climbing *first choice*

The first algorithm to be used in optimization is Hill Climbing first choice.

3.1 Implementation details

Data Files

In a text file there is the information about the problem instance to optimize. The file has three important elements (in this order):
- Maximum number of iterations;
- Number of vertices of the problem;
- Adjacency matrix. For a problem with N vertices, the adjacency matrix has dimension N × N. The value '1' at the position [i, j] indicates that there is a connection between node **i** and node **j**.

Next we present as example of a file to an instance with 10 vertices:

- The algorithm 100 must perform iterations.
- The graph has 10 nodes

- Fist line of the matrix says that the vertex 1 is connected to vertices 2, 3, 4, 5, 6, 9 and 10.

```
100

10

0       1       1       1       1       1       0       0       1       1
1       0       1       1       1       1       1       0       0       0
1       1       0       1       1       1       1       1       0       0
1       1       1       0       1       1       1       1       1       0
1       1       1       1       0       1       1       1       1       1
1       1       1       1       1       0       1       1       1       0
0       1       1       1       1       1       0       1       1       0
0       0       1       1       1       1       1       0       1       0
1       0       0       1       1       1       1       1       0       0
1       0       0       0       1       0       0       0       0       0
```

At the beginning of the execution, the program access to a file like this and gets all the information for the problem to optimize.

Source Code:
The project in C has four modules:

- **main.c**: Contains the function that controls the overall operation of the program. Makes calls to:
  - the algorithm of the application preparation functions (data reading, generation of initial solution);
  - the hill climbing algorithm
  - writes the final results on the monitor;
- **functions.c**: Contains the function that evaluates the quality of a solution.
- **algorithm.c:** contains the function of the ***hill climbing algorithm*** and the function that generates a neighbor of a given solution. It also contains the *skeleton* of the ***simulated annealing algorithm*** to be used in next class.
- **auxiliary.c:** contains some auxiliary functions: generation of random numbers, the generation of the initial solution and reading of the data file

Comments about the implementation

- The data filename must be specified as the first command-line argument. If there are no arguments, you must specify the file name at the beginning of the program;
- The number of vertices of the problem to be solved is obtained by ***init_data()*** function through the file reading and passed as an argument to **hill_climbing()** function;
- The adjacency matrix is filled in ***init_data()*** function and passed as argument to the **hill_climbing()** function;
- In **hill_climbing()** function, the **sol** variable is an array that stores the best solution found so far. The new solution obtained in the neighborhood is stored in **new_sol** variable. If, after evaluation this solution is better, it becomes the current solution for next iteration.
- The space occupied by the arrays that store the solutions and the adjacency matrix is required dynamically.

Conducting experiments with probabilistic algorithms

The hill climbing first-choice algorithm is a probabilistic method. So, a single execution of the algorithm does not obtain statistically valid results. Without these results it is not possible to evaluate the efficacy of algorithm. Whenever an algorithm contains probabilistic components, it is obligatory to carry out several repetitions of the same configuration. The minimum is 10 repetitions, but 30 is the desired value;
When comparing two probabilistic algorithms (or two different settings for the same algorithm), there are two key performance measures:
- Best solution obtained: absolute quality the best solution found;
- Average of the best solutions found (Mean best fitness): average obtained from the best solutions found in each of the repetitions.
In the source code available in Moodle, the number of repetitions can be provided to the algorithm using the command line the number of repetitions to perform (2nd second argument of the command line). If this argument is not specified, will be held 10 repetitions (default setting).

4. Tasks to perform in Class 7

After analyzing the source code provided in Moodle, perform the following steps:

4.1 Experiments

Perform some experiments with the test files available: graph_20.txt, graph_50.txt, graph_100.txt, graph_150.txt and graph_250.txt. The graphs stored on those files have 20, 50, 100, 150 and 250 vertices, respectively.
Complete a table (see enclosed **results.xlsx** file) for each of the test instances. You can analyze the influence that the maximum number of iterations has on algorithm performance. Try to justify any differences in results.

4.2 Accept solution with equal cost

Change the code in order to accept neighbors with the same cost. Perform some experiments with the graphs 100 and 150 vertices and check for changes in the results.

4.3 New Neighborhood

Implement a new function **create_neighbor2** with a new neighborhood for the representation used. Call this function in the hill-climbing algorithm. Repeat the experiments in section 4.1. Analyze any differences compared to previous results.

TASK FOR CLASS 8

Simulated Annealing

Complete a ***simulated annealing*** algorithm for this problem.
Run some experiments with different options for the components of the algorithm: TMAX, TMIN, cooling function, number of iterations of the inner loop (k). Save the results in the file **result.xls**
Compare, analyze and explain the results with the ones obtained by hill-climbing algorithm.

Tip: you can use the function `rand_01()` implemented in `auxiliary.h` to decide if a neighbor is accepted as the new solution.